1. 3所有在DOS操作django都是需要打开E:\pythonProject\djangoProject> 既是在pycharm中创建的文件目录下面进行操作。
2. 操作python的所有DOS命令行开头都是python manage.py

生成一个app 语句：

python manage.py startapp APPNAME

1. 设计一个模型类 ：

使用时要导入django.db.models

将生成的app名称写入settle的INSTALLED\_APPS中；

2.在应用里面的models里面创建类(图书类)

class BookInfo1 (models.Model):

类属性

属性名 = models.字段类型(选项)

Btitle = models.CharField(max\_length=20)

#指定字符串的最大长度

3.根据模型类生成表

1）生成迁移文件

命令：python manage.py makemigrations

2)执行迁移生成表

命令：python manage.py migrate

生成的表在sqlitestidio中找到项目里的db.sqlite3打开即可找到

1. 通过模型类操作数据表
2. 进入项目shell命令：

Python manage.py shell

推出shell命令：quit();

1. 把创建的类导入

如：from booktest1.models import BookInfo1

1. 创建一个是实例对象

如：b=BookInfo1()

之后再对该对象的属性进行赋值操作

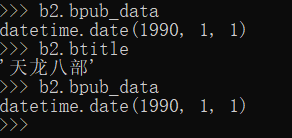
如：b.btitle = '天龙八部'

>>> from datetime import date (导入日期类的包)

>>> b.bpub\_data = date(1990,1,1)

>>> b.save()(保存以上数据到数据中对应的表)

1. 完成操作到sqilitestidio即可查看该实例对象的数据
2. 查询该数据
3. b2= BookInfo1.objects.get(id=1)
4. 变量=类名. Object.get()
5. type(变量)：查询数据类型
6. 获得该id=1数据的每一项的内容

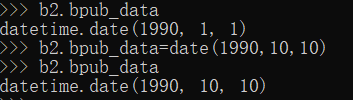


1. 更改该数据
2. 先获得该一条数据的全部内容

即：b2= BookInfo1.objects.get(id=1)

1. 更改数据

b2.bpub\_data=date(1990,10,10)



注意：这么改了之后数据库的内容还是未变，只是改了该实例对象(b2)的值，若要该案数据库里面该项的值，需要再执行一次b2.save()！

注意：第一次创建的对象[b=BookInfo1()

]后保存[b.save()]是插入某一条数据到数据库的表中，第二次调用对象[b2= BookInfo1.objects.get(id=1)]是直接对应表里面的某一条数据，在进行保存[b2.save()]的话是对数据的一个更新

1. 删掉数据
   1. 先获得该一条数据的全部内容

即：b2= BookInfo1.objects.get(id=1)

* 1. b2.delete()

注意：删除数据不需要保存[b2.save()]也可以

1. 增加一个类，实现一对多的关系
2. 现在models里面定义一个类(英雄类)

如：class HeroInfo1(models.Model)

1. 除了添加该类特有的属性之后，还要建立于另一个类的联系，以实现一对多的关系

如：

hbook=models.ForeignKey('BookInfo1',on\_delete=models.DO\_NOTHING)注意:其后参数为所关联的类名

以此来建立与某个类的连接，连接后该类(HeroInfo1)对象可以有多个对应被链接的类('BookInfo1')的对象。

1. 建立对象之间的一对多连接

创建该类(HeroInfo1)的一个实例对象[h = HeroInfo1()]之后，定义该类的参数，之后进行连接h.hbook = b(被连接类BookInfo1的实例对象)，之后进行保存h.save()

注意：因为是一对多的关系所以可以有多个对象对应被连接的类(BookInfo1)的对象，就像一本书里面对应有很多个人物一样。即，可以定义h2.hbook=b.

1. 查询
2. 由一查多：

b.heroinfo\_set\_all(),即可查询到该对象(BookInfo1)所连接的多个类对象(HeroInfo1)(以列表形式实现)



1. 由多查一：

h.hbook，即可查询到该对象所连接的唯一的类。

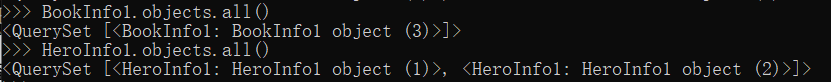


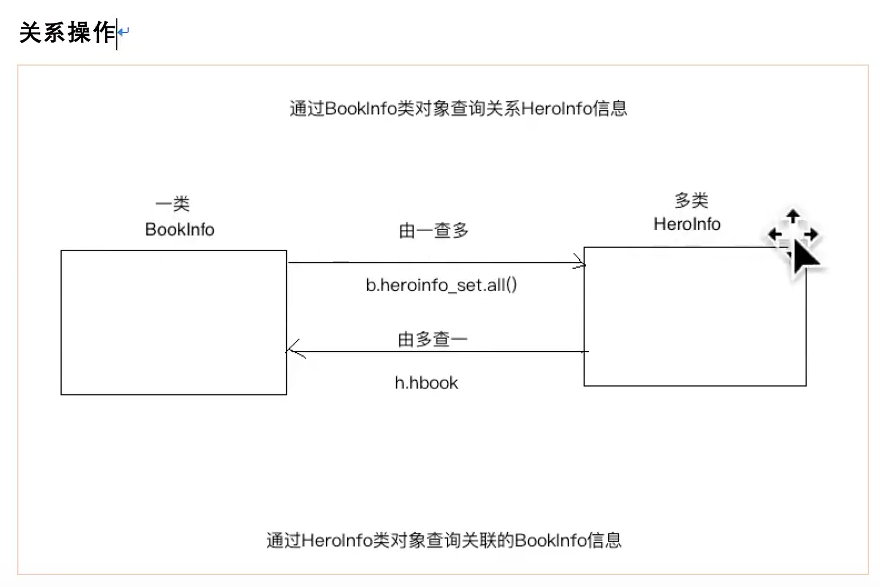
1. 查询表里有多少数据

类名.objects.all()

如：BookInfo1.objects.all()

HeroInfo1.objects.all()





1. 实现后台管理(本地化与创建管理员执行一次即可)
2. 本地化(语言和时区)

在项目的settings.py,里面的

LANGUAGE\_CODE = ‘en-us’

(默认英文，如果‘改成‘zh-hans’则表明使用中文)

TIME\_ZONE = ‘UTC’

(默认英文，如果‘改成‘Asia/Shanghai’则表明使用中国时间)

1. 登陆页面需要管理员账户，则需要创建一个管理员，帮助管理数据库里面的表

命令行：python manage.py createsuperuser

创建成功后运行命令行：

python manage.py runserver

打开http://127.0.0.1:8000/admin 即可以进入管理页面。

1. 注册模型类(一个类只可以注册一次！)，告诉django框架，根据models类生成对应表管理页面。

在app(boooktest1)的admin.py中导入models类的包

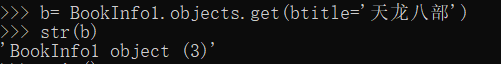
from boooktest1.models import BookInfo1

然后进行模型类的注册(括号里面为操控表的对应模型类名)：

admin.site.register(BookInfo1)

得到的管理页面如图：





页面显示的和表中某一对象的名字[str(b)]一样,所以如果要显示该对象的内容则需要在models类中在admin中已经被注册的类[BookInfo1]中重写str(b)函数，重写如下：

def \_\_str\_\_ (self)

return self.btitle

则显示结果为：



1. 在管理员页面页面对数据库中的表进行修改

进入<http://127.0.0.1:8000/admin>页面之后便可对数据库中的表的数据进行增、删、改、查。

1. 自定义管理页面，控制页面上方要显示表中的哪些内容

自定义模型管理类(在引用的admin中)。模型管理类就是告诉django在生成的管理页面上显示哪些内容。

如：

class BookInfo1Admin(admin.ModelAdmin):

'''图书模型管理类'''

list\_display = ['id','btitle','bpub\_data'](前面必须为list\_display更改无效)

然后在admin注册的模型类后面，加上该类对应的管理类。如：

admin.site.register(BookInfo1,BookInfo1Admin)

1. 视图：在浏览器请求一个页面时，视图函数用于处理请求，并且返回给浏览器页面的内容
2. 定义视图函数(在view.py中)

def index(request):

#进行处理，通过M和T进行交互，与T产生页面，与M操作数据库，与浏览器返回并接受

return HttpResponse('hello python ')

返回HttpResponse对象。Hello python就是返回给浏览器显示的内容。

1. 进行url的配置(使得浏览器输入的url地址和view中处理函数对应起来)：
2. 在应用app(boooktest1)的中新建一个python file然后命名为urls。在其中通过一个列表：

urlpatterns = [

#通过url函数设置url路由配置项,(括号的index/里面可以去掉)

path('index/',views.index),#建立index和视图index之间的关系

【也可以：url(r'index/$',views.index),#是严格匹配，当有多个视图函数对应的多个url中有部分相同时，需要均加入符号$，进行严格匹配，否则每个url都会显示第一个url配置的内容！】

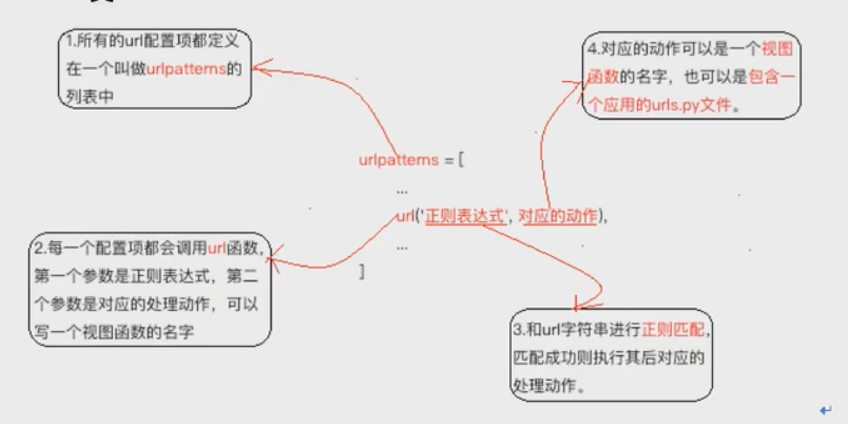
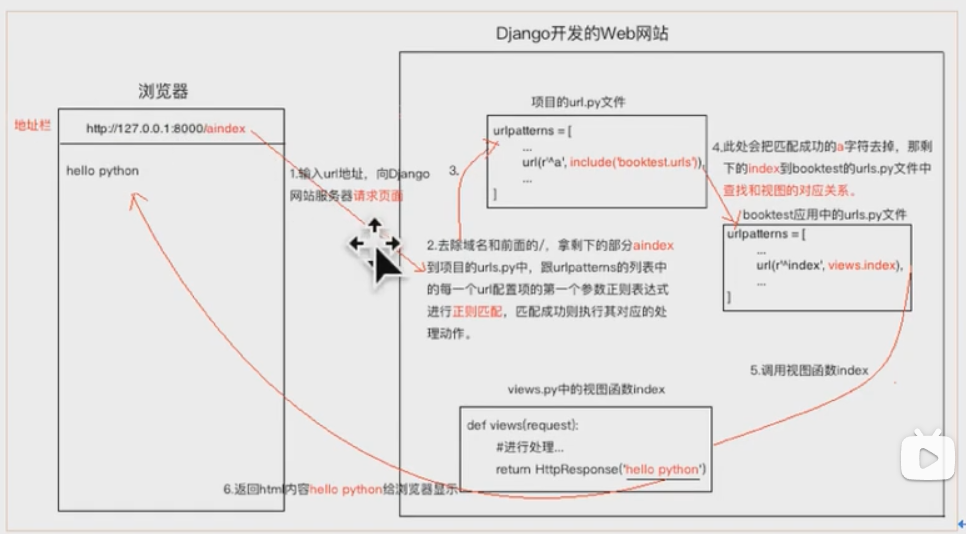
]来配置地址(url)与视图函数(view中的)的对应关系。注：跟[urls.py](https://link.jianshu.com?t=http%3A%2F%2Furls.py)针对APP配置的URL名称，是该APP所有URL的总路径，所以需要在地址栏中输入两个Index，即[http://127.0.0.1:8000/index/index/](https://link.jianshu.com?t=http%3A%2F%2F127.0.0.1%3A8000%2Findex%2Findex%2F" \t "_blank)，这样才可以得到和第一种方法一致的结果。如果不想要两个Index后缀，可以修改myapp下面的url,把path中的index/去掉：

1. 在项目的文件夹(djangoProject)中的urls文件的列表urlpatterns中(即在path—admin的下面)引入以上创建的urls：

path('index/',include('boooktest1.urls')),

【对应上面，也可以：url(r'^',include('boooktest1.urls')),】

#包含boooktest1应用中的urls文件

1. 
2. 
3. 模板：模板文件不仅仅是一个html文件,在模板里面可以写编程语言代码
4. 创建模板文件夹，在项目的根目录下面创建一个命名为templates的文件夹，用来存放模板文件(htmp)，并在项目的settings文件下面的TEMPLATES 里面的'DIRS':写上：[BASE\_DIR / 'templates']或者[os.path.join(BASE\_DIR , 'templates')](项目目录的绝对路径)，一般是自动生成。
5. 在templates的文件夹中新建模板文件(django项目中不止一个模块，一个模块使用一个应用，一个模板属于一个应用)，在其中新建文件夹命名为应用的名字(boooktest1),然后再新建视图函数对应的模板文件(html文件，一般以所对应的视图函数的名字命名)。
6. 使用模板文件：

*'''使用模板文件'''*

#1.加载模板文件,然后去模板目录下面获取html文件的内容，得到一个模板对象。

*temp = loader.get\_template(template\_path)*

# 2.定义模板上下文：给模板文件传递数据

*context = RequestContext(request,context\_dict)*

#3.模板渲染:把变量和语句替换掉，产生标准的html内容,返回替换后的内容，即为标准的html文件

*res\_html = temp.render(context=locals())*

*return HttpResponse(res\_html)*

以上散步内容均被封装再render函数中，直接调用和return render(request,template\_path,context\_dict={})得到一个其中html文件，其中request是一个HttpRespons的对象，template\_path是模板文件夹templates下对应的app的模板文件[写相对路径，即’boooktest1/index.html’]，context\_dict是一个字典，默认为空，可不填。

1. 给模板文件(html文件中)传递数据：

模板变量的使用：{{模板变量名}}

模板代码段：{{%代码段%}}

例如：for循环：

<ul>

{% for i in list %}

<li>{{ i }}

{% endfor %}

</ul>

在html文件中<br/>表示换行。

1. 数据库(mysql)基本命令(命令行)：
2. 连接数据库:mysql -uroot -pROOT
3. 查看数据库：show databases**;**(注意要加分号，下同)
4. 创建数据库：create database name charset=utf8**;**

(name为所创建数据库的名字，语句末尾要加分号)

1. 使用数据库：use django328**;**
2. 查看数据库中的表：show tables**;**
3. 增加数据：insert into booktest\_bookinfo(btitle,bpub\_date,bread,bcomment,isDelete) values

('射雕英雄传','1986-5-1',12,34,0),

【对应数据库中表的各列】

1. **进入/退出虚拟环境**：activate/deactivate

命令行进入该地址，之后输入命令则进去/退出虚拟环境(E:\python\pyvirtualenv\python39env\Scripts)

1. 地址重定向：点击页面之后，服务器不返回页面，而是告诉浏览器再去请求其他的url地址（如果再次请求的url地址与原来相同，则显示原来的页面）。

导入包： from django.http HttpResponseRedirect

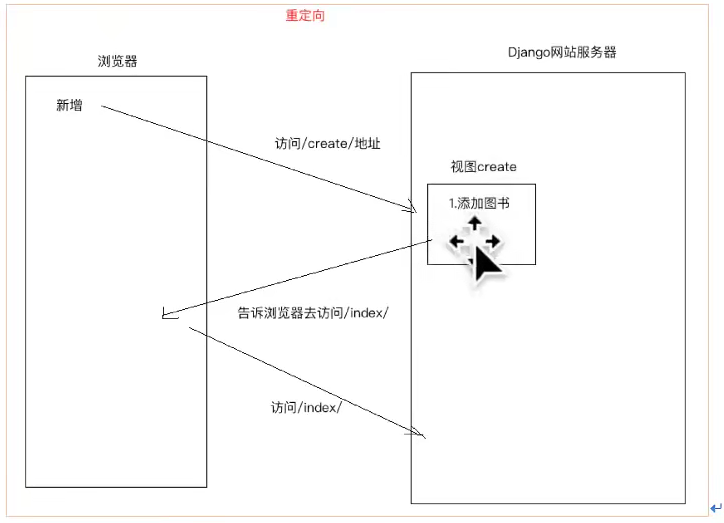
或者：from django.shortcuts import redirect，render

Views函数的返回值：

return HttpResponseRedirect('/index')

return redirect('/index')

返回应答，让浏览器再次访问首页/index/,该过程称为重定向



添加超链接：在html文件中，添加

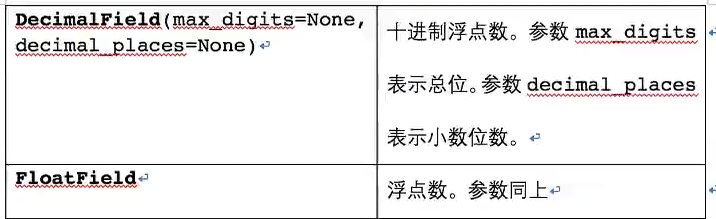
<a href="/creat">title</a>,[/creat表示连接的地址，域名（127.0.0.1：8000） +/creat]

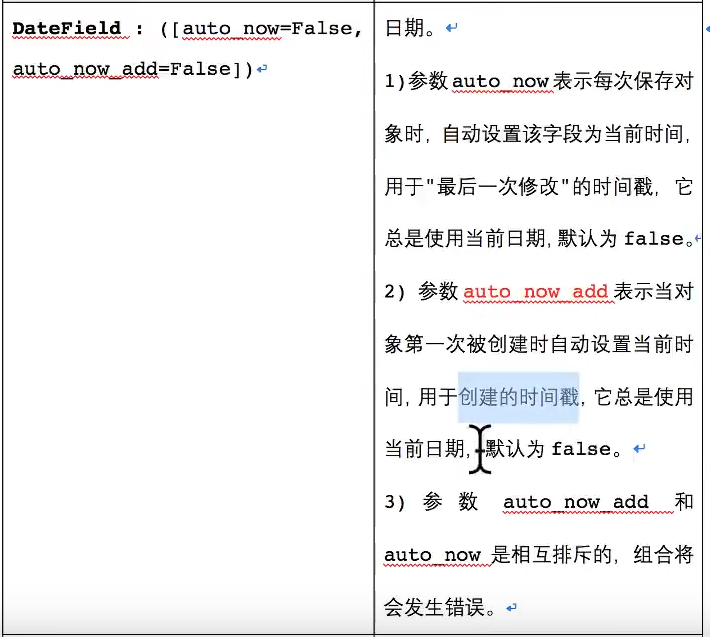
【如果creat前不加/，则会在前面地址的/后面直接拼接creat，如：127.0.0.1：8000/index/creat，则会导致不能访问，所以一般都会加上/】

1. Models的字段属性和选项

字段属性：









选项：





对比：null是数据库范畴的概念，blank是后台管理页面表单验证范畴。利用在类里面利用定义一个Meta类使用db\_table=’表名’可不用执行迁移！

经验：

当修改模型类之后，如果添加的选项不影响表的结构，则不需要重新迁移，商品选项中的default和blank不影响表的结构。

1. 查询

修改mysql的日志文件：

让其产生mysql.log,即是mysql的日志文件，里面记录对mysql数据库的操作记录。

1. 打开mysql实时查询操作（开启general\_log）：

连接数据库：

查询当前开关状态：show variables like "general\_log%";

开启：set global general\_log = ON;

关闭：set global general\_log = OFF;

1. 查询函数：

通过模型类.objects属性可以调用如下函数，实现对模型类对应数据表的查询。







其中，get，filter，exclude都可以写查询的条件

格式为：模型类属性名\_\_条件名=值。

**Filter方法示例：**

1. 判等 条件名 ：exact

例如：查询编号为1的图书。

BookInfo.objects.get(id\_\_exact=1)[exact一般可以省略]

1. 模糊查询

例如：查询书名包含传的图书。contains



例如：查询书以部结尾/开头的图书。endswith/startswith



1. 空查询isnull

例：查询书名不为空的图书。isnull

数据库语句：select \*from booktest\_bookinfo where btitle is not null

命令行语句：BookInfo.objects.filter(btitle\_\_isnull==False)

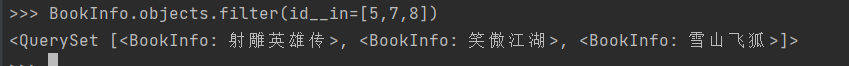


1. 范围查询 in

例如：查询id为1或3或5的图书

数据库语句：select \*from booktest\_bookinfo where id in (1,3,5);

命令行：BookInfo.objects.filter(id\_\_in=[5,7,8])



1. 比较查询：

gt（greate than）大于、lt（less than）小于、gte(equal)大于等于、lte小于等于

例如：查询图书编号大于6的图书

select \*from booktest\_bookinfo where id>6;

命令行：BookInfo.objects.filter(id\_\_gt=6)



1. 日期查询

例如：查询1987年发表的图书

BookInfo.objects.filter(bpub\_date\_\_year=1980)



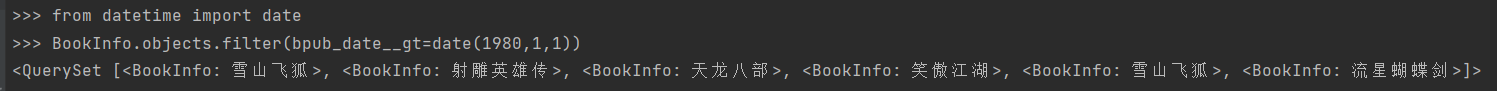
例如：查询11号发表的图书



例如查询1980.1.1后发表的图书

from datetime import date

BookInfo.objects.filter(bpub\_date\_\_gt=date(1980,1,1))



**Exclude方法示例：**

例如：查询id不等于5的图书信息

BookInfo.objects.exclude(id=5)



**Order\_by方法示例：**

作用：对查询结果进行排序

例如：查询所有图书信息，按照id从小到大排序

BookInfo.objects.all().order\_by(‘id’)[.all()可以省略]

BookInfo.objects.order\_by(‘id’)



例如：查询所有图书信息，按照id从大到小进行排序（默认升序）

BookInfo.objects.all().order\_by(‘-id’)



例如：把id>3的图书信息按阅读量从大到小排序显示

BookInfo.objects.filter(id\_\_gt=6).order\_by('-id')



1. F 对象

作用：用于**类属性**之间的额比较

使用之前需要先导入：

from django.db.models import F

例如：查询图书阅读量大于评论量的图书信息

BookInfo. objects.filter(bread\_\_gt=F('bcomment'))

（）后代表阅读量大于评论量的条件



例如：查询图书阅读量大于2倍评论量的图书信息

BookInfo. objects.filter(bread\_\_gt=F('bcomment')\*2)



1. Q对象

作用：对于查询时条件之间的**逻辑关系**。Not and or，可以对Q对象进行与、或、非的操作

使用之前需要先导入：

from django.db.models import Q

例如：查询id大于3且阅读量大于30的图书信息

BookInfo.objects.filter(id\_\_gt=3,bread\_\_gt=30)



BookInfo.objects.filter(Q(id\_\_gt=3)&Q(bread\_\_gt=30))



例如：查询id大于3或者阅读量大于30的图书信息

BookInfo.objects.filter(Q(id\_\_gt=3)|Q(bread\_\_gt=30))



例如：查询id不等于3的图书的信息

BookInfo.objects.filter(~Q(id=3))



1. 聚合函数

作用：对查询结果进行聚合操作

sum count avg max min

1. **aggregate:调用这个函数来使用聚合。**返回值是一个字典

**使用前需要先导入聚合类：**

from django.db.models import Sum,Count,Max,Min,Avg

例如：查询所有图书的数目

BookInfo.objects.aggregate(Count('id'))



例如：查询所有图书阅读量的总和

BookInfo.objects.all().aggregate(Sum('bread'))



1. **count函数** 返回值是一个数字

作用：统计满足条件数据的数目

例如：统计所有图书的数目

BookInfo.objects.count()



例如：统计id>5的所有图书的数目

BookInfo.objects.filter(id\_\_gt=5).count()



总结：



1. 查询集

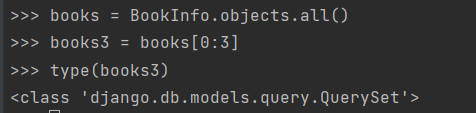
All,filter,exclude,order\_by调用这些函数会产生一个查询集，QuerySet类对象可以继续调用上面的所有函数。

**查询集特性：**

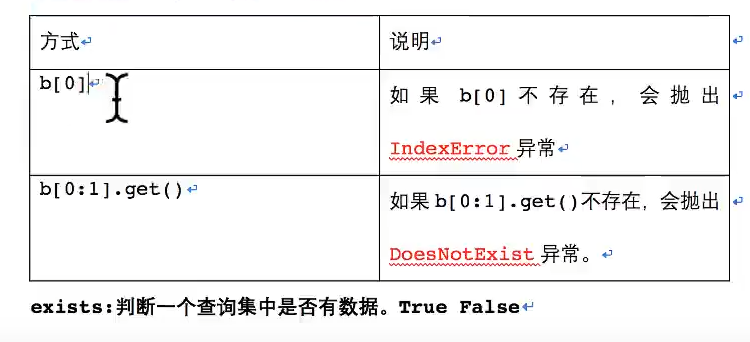
1. 惰性查询：只有在实际使用查询集中的数据的时候才会发生对数据库真正的查询。
2. 缓存：当使用的是同一个查询集时，第一次的时候会发生实际数据库的查询，然后把结果缓存起来，之后再使用这个查询集时，使用的是缓存的结果。
3. 限制查询集：

可以对一个查询集进行取下标或者切片操作来限制查询集的结果。

对一个查询集进行切片操作会产生一个新的查询集，下标不允许为负数。

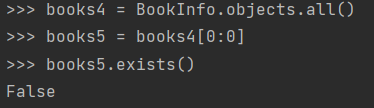


取出查询集第一条数据的两种方式：









1. 模型类关系
2. 一对多关系

例如：图书类-英雄类

models.ForeignKey() 定义在多的类中

1. 多对多关系

例如：新闻类-新闻类型类 体育新闻 国际新闻

models.ManyToManyField() 定义在哪个类中都可以

1. 一对一关系

例如：员工基本信息类-员工详细信息类。员工工号

1. 关联查询（一对多）